CPSC 340 Assignment 6 (due Wednesday December 7 at 11:55pm)

Name(s) and Student ID(s):

1 Sparse Latent-Factor Models

If run the script *example_faces.jl* it will load a set of face images under different lighting conditions. It will then fit a PCA model (with $k = 16$) and creates 3 plots:

- Random faces taken from the dataset (*facesOriginal*).
- Reconstructed versions of these faces based on the PCA model (*facesCompressed*).
- The average face (*faceMu*).
- The principal compoenents (*eigenfaces*).

Since Plots.jl seems to only allow you to show one plot at at time, I think you can type 'facesOriginal" in the REPL to switch to that plot (for example). The reconstructions tend to be reasonable given that they compress each 1024-pixel face down to just 10 numbers, although they often lack specific details and are less accurate for faces that do not look like the average face. In this model, some of the principal components are interpretable (for example, they seem to reflect different lighting conditions), but many of them are not. You can change the value of k in this script to see the effect of including/excluding principal components, if you increase k it will do a better job of reconstructing faces that do not look like the average face.

1.1 Uniqueness of Principal Components

On different runs of the script, you may get different principal components, even though all that changes between runs is the order of the training examples. What is the specific difference between the principal components that are obtained between different runs of the algorithm?

1.2 Non-Negative Matrix Factorization

If you replace the function call to *PCA* with *PCA_gradient*, then instead of using the SVD to compute the principal components it will compute them numerically by using gradient descent steps (alternating between updating W and Z). Note that this returns different principal components because it does not enforce any constraints on W, but it will give an equivalent predictions (up to numerical accuracies).

A disadvantage of PCA is that the principal components tend to be dense (they are all non-zero), which can make them more difficult to interpret. One of the first sparse variations on PCA is non-negative matrix factorization, where we use the PCA objective but add non-negative constraints on W. Using *PCA_gradient* as a template, write a function *NMF* that implements the non-negative matrix factorization (NMF) model.

Hint: you need to make several changes. First, you need to remove the step that centers the data matrix X (NMF cannot model the negative values that get introduced by centering). Second, you need to change the initialization so that negative values of W and Z are set to 0. Third, you need to change the updates of W and Z to use an optimization method that includes the non-negativity constraint. You can use the function *findMinNN* which minimizes a differentiable function subject to non-negative constraints. You will also need to update the *compress* function so that it does not center the data and it finds the non-negative Z minimizing the objective with W fixed, and the *expand* function so that it does not "un-center" the data.

1. Hand in the 3 updated functions required to implement NMF instead of PCA.

2. What do you think the NMF factor represents if you set $k = 1$? (Hint: look at all the plots made by the original script.)
3. What do the NMF factors represent if you set $k = 2$?
4. Hand in a plot of the NMF factors with $k = 16$.
5. Explain why the NMF model should have a higher or lower loss than the PCA model.

2 Neural Networks

2.1 Neural Networks by Hand

Suppose that we train a neural network with sigmoid activations and one hidden layer and obtain the following parameters (assume that we don't use any bias variables):

$$W = \begin{bmatrix} -2 & 2 & -1 \\ 1 & -2 & 0 \end{bmatrix}, v = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

Assuming that we are doing regression, for a training example with features $(x^i)^T = \begin{bmatrix} -3 & -2 & 2 \end{bmatrix}$ what are the values in this network of the hidden units z^i, activations $a^i = h(z^i)$, and prediction $\hat{y}^i$?

2.2 Neural Network Tuning - Regression

The file `example_nnet.jl` runs a stochastic gradient method to train a neural network on the *basisData* dataset, and outputs a plot of the result (depending on your configuration, it may output these plots as the algorithm is running too so that can see how the learning proceeds). Unfortunately, in its current form the neural network does not fit the data well. Modify the training procedure and model to improve the performance of the neural network. Hand in your plot after changing the code to have better performance, and list the changes you made.

Hint: there are many possible strategies you could take to improve performance. Below are some suggestions, but note that the some will be more effective than others:

- Changing the network structure (*nHidden* is a vector giving the number of hidden units in each layer).
- Changing the training procedure (you can change the stochastic gradient step-size, use decreasing step sizes, use mini-batches, run it for more iterations, add momentum, switch to *findMin*, use Adam, and so on).
- Transform the data by standardizing the feautres, standardizing the targets, and so on.
- Add regularization (L2-regularization, L1-regularization, dropout, and so on).
- Change the initialization.
- Add bias variables.
- Change the loss function or the non-linearities (right now it uses squared error and tanh to introduce non-linearity).
- Use mini-batches of data, possibly with batch normalization.

2.3 Neural Network Tuning - Classification

The file `example_mnist35_logistic` runs a stochastic gradient method to train a neural network on the '3' and ''5' images from the MNIST dataset, which obtains a test error of 0.03. If you instead run `example_mnist35_nnet.jl`, it fits a neural network with stochastic gradient descent on the same data (using the squared error for training and classifies by taking the sign of the prediction) but obtains a worse error. Modify the neural network training procedure and model so that the neural network has a better error on the test set than the 0.03 achieved by logistic regression. List the changes you made and the best test performance that you were able to obtain.

3 Using a Deep Learning Package

The script *example_flux.jl* shows how the compute the loss function and gradient of a neural network with one hidden layer (with 3 units) using a variety of strategies, and how to update the parameters based on an SGD update using a variety of strategies. This includes the backpropagation code from the previous question, as well automatic differentiation and other functionality provided by the *Flux* package. The function does not produce any output, but you can verify that all methods return the same results up to numerical precision.

1. In the previous assignment we concatenated all parameters into one big vector of parameters w. What is an advantage of the other approaches in the script, where we have a matrix W and a vector v?
2. If we use use `Dense(W)` for the first layer in the network, how many parameters does the first layer have and what do the extra parameters represent? Hint: you can use `params(layer)` to get the parameters of a layer.
3. Give code for implementing a neural network with 2 hidden layers, using the `Dense` function within the `Chain` function.
4. It we use *Conv((5,5),3=>6,relu)* for a layer, why do we get a layer with 456 parameters?

5. Re-write the model and training procedure you developed for Question 2.3 to use one of these alternate ways to compute the gradient and the update of the parameters (so you should not be calling *NeuralNet.jl* anymore). Re-state the changes you made, hand in your modified code, and report whether you noticed any performance differences (in terms of runtime, memory, or accuracy).

4 Very-Short Answer Questions

1. Consider fitting a linear latent-factor model, using L1-regularization of the w_c values and L2-regularization of the z_i values,

$$f(Z,W) = \frac{1}{2}\|ZW - X\|_F^2 + \lambda_W \sum_{c=1}^{k} [\|w_c\|_1] + \frac{\lambda_Z}{2} \sum_{i=1}^{n} \left[\|z_i\|^2\right],$$

   (a) What is the effect of λ_Z on the two parts of the fundamental trade-off in machine learning?

   (b) What is the effect of k on the two parts?

   (c) Would either of answers to the *previous two questions* change if $\lambda_W = 0$?

2. Which is better for recommending movies to a new user, collaborative filtering or content-based filtering? Briefly justify your answer.

3. Consider using a fully-connected neural network for 3-class classification on a problem with $d = 10$. If the network has one hidden layer of size $k = 100$, how many parameters (including biases) does the network have?

4. Consider fitting a neural network with one hidden layer. Would we call this a parameteric or a non-parametric model if the hidden layer had n units (where n is the number of training examples)? Would it be parametric or non-parametric if it had d^2 units (where d is the number of input features).

5. Describe an experimental setup where you might see a double descent curve when fitting a logistic regression model with L2-regularization. You can assume you have a way to generate new relevant features. Caution: the global minimum is unique in this model so the double descent would not come from choosing among multiple global minima.

6. Consder a neural network with L hidden layers, where each layer has at most k hidden units and we have c labels in the final layer (and the number of features is d). What is the cost of prediction with this network?

7. What is the cost of backpropagation in the previous question?

8. What is the "vanishing gradient" problem with neural networks based on sigmoid non-linearities?

9. What is one advantage of using automatic differentiation and one disadvantage?

10. Convolutional networks seem like a pain... why not just use regular ("fully connected") neural networks for image classification?